Paper 233-2012

Seamless Reporting Automation through the Integration of JMP®, SAS®, and VBA

Rachel Poulsen, Raghunathan Chakravarthy, TiVo Inc, Alviso, CA, USA

ABSTRACT

The TiVo user interface (UI) is continuously enhanced for design. To monitor and improve UI performance, statistical tests are run on multiple test cases across various dimensions, and weekly reports are generated. The reports require the flexibility of Excel, the analysis power of JMP® and SAS®, and the reporting convenience of PowerPoint. This can be accomplished by integrating JMP, SAS® ODS, and Excel VBA to create an automation tool that seamlessly runs the analyses, conditionally formats the table output, and generates reports in PowerPoint. JMP automates the statistical analysis, SAS® ODS calls Excel VBA macros for the conditional formatting of Excel data sets, and SAS® ODS generates reports in PowerPoint. This paper discusses the tasks involved in the process flow of the automation tool.

INTRODUCTION

TiVo is a digital video recorder (DVR) that provides users with an on-screen user interface of scheduled television programming, current recordings, online video and music content, and a database of related television and movie content. TiVo's User Interface (UI) provides a range of features when it is connected to a home network, including video on demand, advanced search, personal photo viewing, music offerings, and broadband video content. The TiVo UI consists of multiple screens allowing the user to explore content and interact with various applications. The continued enhancement of the UI requires ongoing monitoring of the performance. TiVo defines performance as the responsiveness of the UI while interacting with the software. Performance is measured by collecting the time it takes to perform a core list of tasks within the UI. As the UI is enhanced, it is critical that performance of these core tasks meets the required level of satisfaction and does not degrade.

Not all statistical analyses are intended to be ongoing reports. Oftentimes an initial request for a one time analysis evolves into a need for ongoing support. In these instances the limitations of the statistical program used for the analysis can be frustrating and may require redoing the entire analysis. However, the limitations of JMP® and SAS® can be overcome using the strengths of other programs through integrated code. JMP® can borrow the power of SAS® ODS and the convenience of Excel VBA to create conditionally formatted Excel tables and PowerPoint slides. Combining the power of JMP®, SAS®, and Excel VBA, the reporting possibilities are unrestricted, various business needs can be accommodated, and automation is seamless. This paper will discuss step by step how to turn a simple one-time JMP® analysis into an ongoing automated report in PowerPoint and Excel using the integration of SAS® and Excel VBA.

DATA AND METHODOLOGY

The data consist of a repeated measures experiment involving two factors measured against a list of core performance tasks. The initial dataset was collected by running and measuring each task over 30 times each on two different software (SW) versions: the released SW version and the candidate SW version. All other factors were controlled for where possible. The tests were run using the same measurement tool, hardware, server, and configuration. The initial request was to compare the release candidate software version to the production software version and to identify any possible regressions in performance across the performance tasks.

Each of the performance tasks were analyzed using a one-way analysis and power analysis in JMP®. The final results of each comparison were placed into Excel tables, along with some basic descriptive statistics and details about the data collection. Results were placed in Excel tables for the convenience and editing purposes. The recipients of the report wanted the ability to sort and filter the tables by different variables, and to present only a subset of the tables in executive reviews. The final tables were also placed into PowerPoint along with some graphs and charts summarizing the results at a high level. PowerPoint presentations were not necessary for this request, but were chosen for their familiarity.

After the initial analysis, ongoing performance measurement across multiple projects was set up and weekly analysis and reporting was requested for each project. For ongoing analyses, it is useful to save analysis scripts in JMP®. Saving scripts allows for the saved analysis to be run on new datasets with the click of a button. JMP® journals and windows are useful for organizing and presenting analysis reports. However, this requires the end user to have a license to JMP®. JMP® journals can be exported in Word, HTML, or PDF format, but the final tables have formatting limitations. Rows, columns, or specific observations cannot be highlighted in JMP® output. It is also difficult to select a subset of the final tables or filter by specific values. With many performance tasks measured on over multiple projects, it is critical to easily identify the performance tasks with statistically significant differences. It is also critical to be able to filter the table to easily identify problem areas. The initial analysis highlighted these tasks in Excel workbooks, but an automated analysis script in JMP® would result in reporting limitations.

Automating and outputting the Excel tables in the final reports is trivial using Excel tagsets in SAS® ODS. Traffic-lighting in SAS® ODS can export conditionally formatted tables, satisfying the need for highlighted specific values in the final tables. Excel tagsets in SAS® ODS can also be used to format the final Excel tables for nicer presentation. With the initial analysis completed and saved in JMP®, moving the analysis to SAS® would require re-doing the entire analysis. JMP® and SAS® can be integrated using the *SAS Submit* command in JMP® scripting language (JSL). With the analysis script already saved, SAS® ODS code can be added to the end of the JMP® script to format the final tables in Excel, and no redundant work is required.

Automating and outputting the final charts and tables into PowerPoint is not a trivial task. Once the final charts and graphs are created using JMP® and SAS® integration, VB scripting can be used to upload the graphics to PowerPoint and VBA macros can be used to copy and paste Excel tables into PowerPoint. SAS® has the ability to call VBA macros and VB scripts.Another option is to export all tables and charts as PDF documents and upload into a PDF presentation using LaTex. LaTex can also integrate with SAS® code using the LaTex tagset in SAS® ODS. However, converting excel tables into PDF requires additional software. To complete full automation of the reports, a windows batch script was used to call the necessary code. Figure 1 illustrates the analysis to reporting flow.

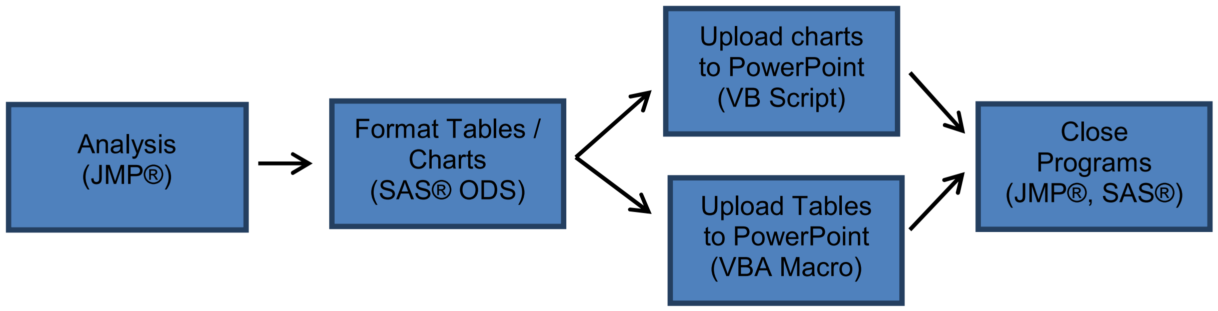

Analysis (JMP®)
Format Tables / Charts (SAS® ODS)
Upload charts to PowerPoint (VB Script)
Upload Tables to PowerPoint (VBA Macro)
Close Programs (JMP®, SAS®)

Figure 1. Automated Reporting Flow

AUTOMATION OF THE JMP® APPLICATION

JMP® was used to complete the original statistical analysis comparing each of the performance test cases measured on the candidate software to the corresponding test case measured on the release software. Once the analysis was created, the analysis script was saved. JMP Scripting Language (JSL) is a convenient way to automate a frequently used analysis. JMP® has the ability to connect to a database, but data are not always available from a database. In this case, data were saved in a delimited text file. JMP® also has the ability to open various file types. The command *Open* in JSL will open a file in a specified location and the command *Pick File* will prompt the user for the location of the data file. The latter was used for this project.

```
current_file=Pick File("Select the File Needed for Performance Analysis", "\Path
for\Directory");
```

One of the drawbacks to storing data outside a database is the increased likelihood of human error. JSL prompts were also created to handle the common human errors for this analysis as well as data issues that would result in termination the automation. These included character values in the measurement column, multiple SW versions in the SW column (when only one should be specified), and missing values. When an error was found the user would be prompted of the error, asked to fix the error, and the JSL script would terminate. The following JSL code gives an example of how one of these prompts can be created.

```
//Check for character values in the Measurement column (due to typos)

coltypecheck=column("Measure")<<Get Modeling Type;
For(i=1, i<=1, i++, //i is just a dummy value for a loop
    if(coltypecheck=="Continuous", Break()); //Break loop if data type correct
    Dialog(
    "Performance data has been rejected because there are non-numeric measurements
in the current data file. Please check the data for typos.",
    Button("OK"), Title("Error")
    );
    Current data table()<<Close window;
    Exit(); //closing all windows and exiting JMP
);
```

Another drawback to automated analysis is the lack of Analyst verification. For example, an Analyst will notice the large outliers or small sample size in a manual analysis. An Analyst will also make extra edits so the final report can be more readable. For example, an Analyst may present the statistical conclusion ("statistically faster") for the t-test as opposed to the t-score. In order to reduce these errors, JSL code was written to check assumptions and also to make the final results more readable. Specifically, a power analysis was completed on the One-way analysis for each test case. If the power is too small (<0.80), the final result is "sample size too small."

```
result={}; //initializing list to store readable results

//t scores for all tests stored in “tscore”
//power for all tests stored in “pwr”
//ntests is the number of test cases

for(i=1, i<=ntests, i++,
    if(tscore[i]>1.96,
           result[i]="Faster",
           tscore[i]<(-1.96),
           result[i]="Slower",
           tscore[i]>(-1.96) & tscore[k]<1.96 & pwr[k]>0.80
           result[i]="Equal",
           result[i]="Sample size too small"
    );
);
```

The results can then be placed in a JMP® table with the 'result' variable as one of the columns.

AUTOMATION OF THE SAS® APPLICATION

After the data are cleaned for common errors, analysis complete, and results output in a readable format, SAS® is used to format the final tables. SAS® code can be written directly in JSL using the *SAS Submit* command. Excel is a useful tool for formatted worksheets that can be easily edited, and is familiar and available to most recipients. Excel tagsets in SAS® ODS are a convenient way to make SAS® output available in Excel. Also convenient for automated formats, is defining a template of common formatting that can then be passed to all ODS output using PROC TEMPLATE. Below is an example of a formatting template that can be applied to all Excel tables created using SAS® ODS.

```
proc template;
    define style styles.MysansPrinter;
           parent = styles.sansPrinter; /*Using sansPrinter style*/
           style myheader from header / background = white
           font_size = 8pt
           font_face = "Arial"
           just = center;
           style mydata from data / background = white
           font_size = 8pt
           font_face = "Arial"
           just = left;
end;
run; quit;
```

For the analysis described in this paper, summary charts were also created in SAS®. Charts were created in SAS® as opposed to JMP® so that the VBA macros could easily access charts saved in the local SAS® directory. JMP® can integrate with SAS® which can integrate with VBA, but JMP® cannot directly call VBA and VBA cannot directly access JMP® graphics. SAS® charts were also saved as .emf files for ease of VB scripting language. SAS® will then open and execute a VB script to upload the document. The VB script can be found in the appendix. The SAS® code used to call the VB script can be seen below.

```
filename rs pipe "cscript //nologo ""\Myfile\path\vbscript.vbs""";
    data _null_;
    infile rs;
    input;
    put _infile_;
    run;
    quit;
```

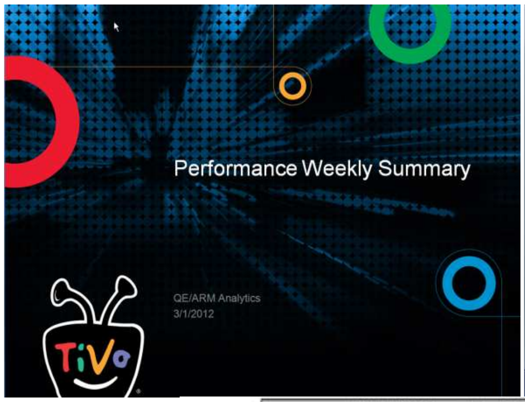
AUTOMATION OF THE VBA APPLICATION

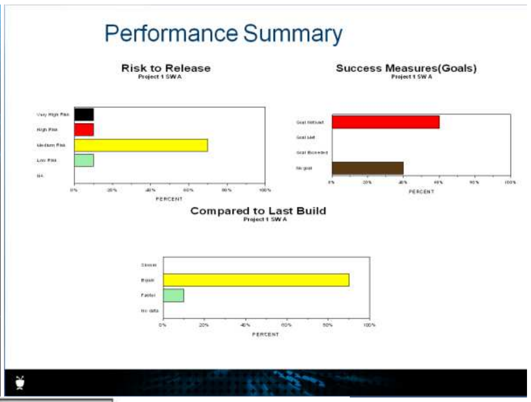
Excel VBA macros were used to automatically upload snapshots of the final tables. Recall that these tables were initially created in JMP® and formatted in SAS® using Excel tagsets in SAS® ODS. SAS® is used to open the Excel application and execute the VBA macros. VBA macros are used to setup the PowerPoint reference, and copy and paste the final formatted Excel tables into PowerPoint slides. SAS® will also execute a final VB script to close the Excel files. The following SAS® code calls the VBA macros that copy and paste Excel tables into PowerPoint. The VBA macro can be found in the appendix.

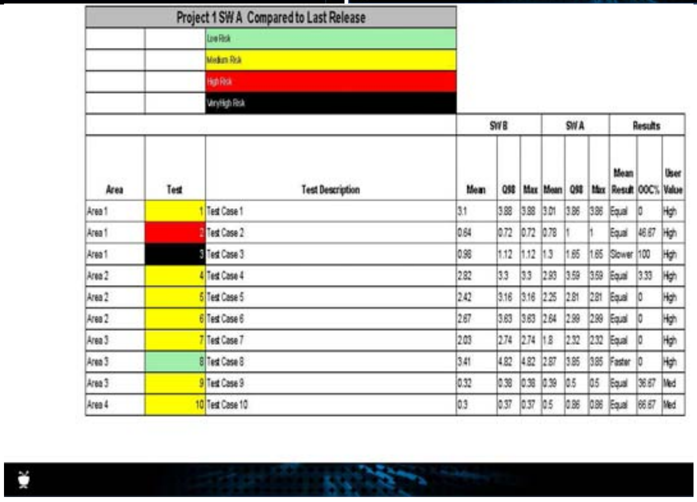
```
 options noxwait noxsync;
    x "Start Excel";
    data _null_;
    rc = sleep(3);
    run;

    filename EXCEL DDE 'EXCEL|SYSTEM';
    options xsync;
    data _null_;
    file excel;
    put '[open("\File Path for VBA Macro\MyMacro.XLSM")]';
    run;

    data _null_;
    file excel;
    put '[RUN("MyMacro")]';
    run;
```

RESULTS

The following are blinded results comparing the performance of a candidate SW build to the production SW build using the end-to-end automation tool. The results were placed into a company PowerPoint template and saved to location accessible to all necessary recipients. The entire automation from JMP® analysis to PowerPoint takes less than one minute to complete for one project. The automation allows for comparisons of the candidate SW version to up to two different SW versions as well as static goals for the performance tasks. The high level SAS ® charts as well as one table created using SAS® ODS are shown below.







Project 1 SW A Compared to Last Release												
		🟩 Low Risk										
		🟨 Medium Risk										
		🟥 High Risk										
		⬛ VeryHigh Risk										
			SW B			SW A			Results			
Area	Test	Test Description	Mean	Q98	Max	Mean	Q98	Max	Mean Result	OOC%	User Value
Area 1	🟨 1	Test Case 1	3.1	3.88	3.88	3.01	3.86	3.86	Equal	0	High
Area 1	🟥 2	Test Case 2	0.64	0.72	0.72	0.78	1	1	Equal	46.67	High
Area 1	⬛ 3	Test Case 3	0.98	1.12	1.12	1.3	1.65	1.65	Slower	100	High
Area 2	🟨 4	Test Case 4	2.82	3.3	3.3	2.93	3.59	3.59	Equal	3.33	High
Area 2	🟨 5	Test Case 5	2.42	3.16	3.16	2.25	2.81	2.81	Equal	0	High
Area 2	🟨 6	Test Case 6	2.67	3.63	3.63	2.64	2.99	2.99	Equal	0	High
Area 3	🟨 7	Test Case 7	2.03	2.74	2.74	1.8	2.32	2.32	Equal	0	High
Area 3	🟩 8	Test Case 8	3.41	4.82	4.82	2.87	3.85	3.85	Faster	0	High
Area 3	🟨 9	Test Case 9	0.32	0.38	0.38	0.39	0.5	0.5	Equal	36.67	Med
Area 4	🟨 10	Test Case 10	0.3	0.37	0.37	0.5	0.86	0.86	Equal	66.67	Med

CONCLUSION

Transforming a one-time analysis into an ongoing report does not require re-doing the analysis with the tools available in JMP® and SAS® to communicate with other SW languages. This paper has illustrated how reporting automation can be seamless through the ability of JMP® to communicate with SAS® and the ability of SAS® to communicate with VBA. Integrated code allows Analysts to focus their efforts on analyzing and away from tedious formatting and copying and pasting into reports.

If PowerPoint is not required for final reports, another option for automated reporting (not discussed in this paper) is LaTex. LaTex creates final presentations in PDF format. LaTex also has the ability to call parameters, allowing for more flexibility in reporting. Reporting titles and labels for slides can be passed as parameters through LaTex coding. LaTex tagsets are also available in SAS® ODS, allowing for easy integration of code.

APPENDIX

VB SCRIPT TO UPLOAD SAS CHARTS INTO POWERPOINT

```
  'Open PPT template
  Set objPPT = CreateObject("PowerPoint.Application")
  Set objPresentation = objPPT.Presentations.Add()
  objPresentation.ApplyTemplate "\File Path to PPT Template\PPT Template.potx"

  'Title Slide
  Set objSlide = objPresentation.Slides.Add(1,1)
  Set objsh = objSlide.Shapes
  Set objTitle = objsh.Item(1)
  objTitle.TextFrame.TextRange.Text="Title"
  Set objTitle = objsh.Item(2)
  objTitle.TextFrame.TextRange.Text="Author1/Author2
  " &Date

  'Create slide with three pictures
  Set objSlide = objPresentation.Slides.Add(3,11)
  Set objsh = objSlide.Shapes
  Set objTitle = objsh.Item(1)
  objTitle.TextFrame.TextRange.Text="            Title for Slide "
  Set pic = objSlide.shapes.AddPicture("\File Path to Picture 1\Picture1.emf",-1,-
  1,42,79,330,230)
  Set pic = objSlide.shapes.AddPicture("\File Path to Picture 2\Picture2.emf",-1,-
  1,400,79,310,230)
  Set pic = objSlide.shapes.AddPicture("\File Path to Picture 3\Picture3.emf",-1,-
  1,186,276,330,230)

  'Save PPT presentation
  objPresentation.SaveAs("\File Path to Save PPT\MYPowerPointPresentation.ppt")
  objPresentation.Close
  objPPT.Quit
```

VBA MACRO TO CONVERT XML FILE TO XLSM FILE

```
Sub SetupMacro()

  Dim strDay: strDay = Day(Date)
  Dim strYear: strYear = Year(Date)
  Dim strHour: strHour = Hour(Time)
  Dim strMin: strMin = Minute(Time)

  'Open excel
  Set xlobj1 = CreateObject("Excel.Application")

  Set FSO = CreateObject("Scripting.FileSystemObject")
  xlobj1.DefaultFilePath = "\File Path\for Output"
  xlobj1.Application.DisplayAlerts = False

  'Open performance analysis output
  xlobj1.Workbooks.Open ("\File Path for table 1\saved as xml doc from SAS ODS Excel
  output\table.xml")

  'Turn off messages and remove defautl xls sheets
  xlobj1.Application.DisplayAlerts = False

  xlobj1.ActiveWorkbook.VBProject.VBComponents.Import "\File path to .bas
  file\Detxltoppt.bas"

  'Save as .xlsm file
```

```
   xlobj1.ActiveWorkbook.SaveAs "\File Path for xlsm output for table 1\table1.xlsm",
   FileFormat:=52
   'xlobj1.Activeworkbook.ExportAsFixedFormat 0, output2 & ".pdf", 0, 1, 0,,,0
   xlobj1.Application.DisplayAlerts = True
   xlobj1.ActiveWorkbook.Close
   xlobj1.Quit

End Sub
```

VBA CODE TO COPY XLSM FILE AND PASTE INTO POWERPOINT

```
   Sub setpptreference()
   Set ID = ThisWorkbook.VBProject.References
       ID.AddFromGuid "{91493440-5A91-11CF-8700-00AA0060263B}", 2, 10

   End Sub

   Sub loadxltoppt()
   Dim PPApp As PowerPoint.Application
   Dim PPPres As PowerPoint.Presentation
   Dim PPSlide As PowerPoint.Slide
   Dim slidecount As Long
   Dim ictr As Integer
   Dim ws As Worksheet

   ' Create instance of PowerPoint
   Dim strPresPath As String, strExcelFilePath As String, strNewPresPath As String
   strPresPath = "\File Path to Saved PPT\MYPowerPointPresentation.ppt "
   Set PPApp = CreateObject("Powerpoint.Application")
   PPApp.Visible = msoTrue
   Set PPPres = PPApp.Presentations.Open(strPresPath)
   ' Set PPPres = PPApp.Presentations.Add

   ' XltoPPT changes begin here
   For Each ws In ActiveWorkbook.Worksheets

   'Dim firstsName  : firstsname = xlobj1.Worksheets( 1 ).Name
       Dim sName: sName = ws.Name
       Worksheets(sName).Activate

       Set ExcelLastCell = ActiveSheet.Cells.SpecialCells(xlLastCell)

       LastRowWithData = ExcelLastCell.Row
       Row = ExcelLastCell.Row
       Do While Application.CountA(ActiveSheet.Rows(Row)) = 0 And Row <> 1
       Row = Row - 1
       Loop
       LastRowWithData = Row ' Row number

       LastColWithData = ExcelLastCell.Column
       Col = ExcelLastCell.Column
       Do While Application.CountA(ActiveSheet.Columns(Col)) = 0 And Col <> 1
       Col = Col - 1
       Loop
       LastColWithData = Col ' Column number
       Range(Cells(1, 1), Cells(LastRowWithData, LastColWithData)).Select

       ' Set a VBE reference to Microsoft PowerPoint Object Library


       ' Make sure a range is selected
       If Not TypeName(Selection) = "Range" Then
```

```
        MsgBox "Please select a worksheet range and try again.", vbExclamation, _
            "No Range Selected"
    Else
        ' Reference existing instance of PowerPoint
        Set PPApp = GetObject(, "Powerpoint.Application")
        ' Reference active presentation
        Set PPPres = PPApp.ActivePresentation
        PPApp.ActiveWindow.ViewType = ppViewSlide
        Application.Wait (Now() + TimeValue("00:00:10"))
        Selection.CopyPicture Appearance:=xlScreen, _
            Format:=xlPicture

        ' Reference active slide
        slidecount = PPPres.slides.Count
        Set PPSlide = PPPres.slides.Add(slidecount + 1, pplayoutblank)
        PPApp.ActiveWindow.View.GoToSlide PPSlide.slideIndex
       ' Set PPSlide =
PPPres.slides(PPApp.ActiveWindow.Selection.SlideRange.slideIndex)

        ' Paste the range
        PPSlide.Shapes.Paste.Select

        With PPApp.ActiveWindow.Selection.ShapeRange
           .LockAspectRatio = msoFalse
           .Left = 85
           .Top = 3
           .Height = 450
           .Width = 626
         End With


        ' Align the pasted range
       ' PPApp.ActiveWindow.Selection.ShapeRange.Align msoAlignCenters, True
       '  PPApp.ActiveWindow.Selection.ShapeRange.Align msoAlignMiddles, True

    End If
Next ws

With PPPres
    .SaveAs strPresPath
    .Close
End With
' Quit PowerPoint

PPApp.Quit

Activeworkbook.close False
' Clean up
Set PPSlide = Nothing
Set PPPres = Nothing
Set PPApp = Nothing


Dim x As Object
n = Application.VBE.ActiveVBProject.References.Count

    Do While Application.VBE.ActiveVBProject.References.Count > 0 And n > 0
        On Error Resume Next
        Set x = Application.VBE.ActiveVBProject.References.Item(n)
        y = x.Name
        If y = "POWERPOINT" Then
            Application.VBE.ActiveVBProject.References.Remove x
        End If
```

```
            n = n - 1
        Loop

    End Sub
```

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Rachel Poulsen
Enterprise: TiVo
Address: 2190 Gold Street
City, State ZIP: Alviso, CA 95002
Work Phone: 408-519-9203
Fax:
E-mail: rpoulsen@tivo.com
Web:

Name: Raghunathan Chakravarthy
Enterprise:
Address:
City, State ZIP:
Work Phone:
Fax:
E-mail: statsraghu@gmail.com
Web: